Лабораторная работа №4

OpenMP

Цель работы: знакомство с основами многопоточного программирования.

Инструментарий: работа должна быть выполнена на C или C++. В отчёте указать язык и компилятор, на котором вы работали. Стандарт OpenMP 2.0.

Порядок выполнения и сдачи работы:

1. Изучить конструкции OpenMP для распараллеливания вычислений.
2. Написать программу, решающую поставленную задачу, любого из вариантов сложности. Принимается решение только 1 задачи.
3. Провести серию экспериментов по измерению времени работы программы. Подробнее: Задание.
4. Оформить отчет в формате pdf.
5. Загрузить файл отчета и файлы с исходным кодом (расширения *.c/*.h/*.cpp/*.hpp) на github в выданный вам репозиторий.
6. Запустить автотесты. Подробнее: Github и Github Actions

Шаблон отчета и порядок отправки работ на проверку

ЭВМ осень 2022. Порядок сдачи лабораторных работ

Содержание отчета

1. Минититульник (таблица с ФИО и названием работы из шаблона).
2. Цель работы и инструментарий.
3. Описание конструкций OpenMP для распараллеливания команд.
4. Описание работы написанного кода.
5. Результат работы написанной программы с указанием процессора, на котором производилось тестирование.

6. Экспериментальная часть (Задание)
7. Список источников.
8. Листинг кода.

Пояснения:

В *пункте 3* опишите какие конструкции вы использовали и кратко как они работают.

Пункт 4. Описание работы программы. Как реализовано распараллеливание кода при помощи ранее описанных конструкций.

В *пункт 5* вставить лог вывода программы на входных параметрах, указанных в задаче.

Пункт 6 должен содержать графики, которые подробно описаны в разделе Задание.

Если вы пользовались какими-то источниками информациями (спецификация, статьи и пр.), то в *пункте 7* нужно оставить ссылки на эти интернет-ресурсы.

Важно:

1. Будет оцениваться как правильность результата, так и скорость работы.
2. Без кода теория не оценивается, поэтому пытаться посылать отчет в таком случае бессмысленно.
3. Плагиат карается отрицательными баллами за всю работу.
4. Стандарт OpenMP 2.0 https://www.openmp.org//wp-content/uploads/cspec20.pdf ([updated 17.12] важно: использовать reduction нельзя, техники оптимизации мы рассмотрели на паре)

Задание

Необходимо написать программу, решающую одну из задач, описанных в разделе Варианты. Чем сложнее вариант, тем больше баллов за работу вы можете получить.

Помимо написания программы, необходимо провести замеры времени работы на вашем компьютере и привести графики времени работы программы (некоторые графики из следующих подпунктов можно объединить в один):

1. при различных значениях числа потоков при одинаковом параметре schedule* (без chunk_size);
2. при одинаковом значении числа потоков при различных параметрах schedule* (с chunk_size);
3. с выключенным openmp и с включенным с 1 потоком.

* `schedule(kind[, chunk_size])` – kind принимает значение static или dynamic, chunk_size – 1 и несколько (1-2) других значений

В каждом варианте результат работы программы выводится в выходной файл, а время работы программы - в поток вывода (stdout). Формат вывода (в формате Си): `“Time (%i thread(s)): %g ms\n”`. Время работы выводится только в консоль. В данном случае имеется в виду время работы алгоритма без времени на считывание данных и вывод результата.

Для минимизации влияния степени загруженности процессора другими процессами, время должно усредняться по нескольким запускам.

Время в программе нужно измерять при помощи `omp_get_wtime()` (раздел 3.3).

Про `schedule` также можно почитать в спецификации (разделы 2.4.1 и Appendix D).

Варианты

Easy

Расчет площади круга методом Монте-Карло.

Во входном файле записаны радиус `r (вещественное положительное)` и количество значений `N (целое положительное)`, которое будет генерироваться.

Пример:

input	output
1 10	2.8
1 100	3.16
1 1000	3.148

Файл, содержащий функцию `main` должен называться `easy.cpp` или `easy.c`. Если этого не сделать, то автотесты не пройдут, т.к. будет непонятно, какой вариант вы выполнили.

Hard

Пороговая фильтрация изображения методом Оцу. Необходимо реализовать алгоритм для трёх порогов. Более подробно про метод можно почитать в Оцу.pdf. Гарантируется, что в изображении будет как минимум 4 разных уровня интенсивности (4 цвета/оттенка серого).

В программе должны выполняться следующие действия (не считая проверок корректности):

1. Чтение входного изображения из файла.
2. Вычисление гистограммы входного изображения.

3. Полный перебор всех комбинаций порогов и выбор наилучшей комбинации среди них.
4. Преобразование входного изображения в новое, где каждый оттенок преобразуется в новый.
5. Запись полученного изображения в выходной файл.

Правило преобразования оттенков:

$$[0 .. f_0] \to 0, [f_0 + 1 .. f_1] \to 84, [f_1 + 1 .. f_2] \to 170, [f_2 + 1 .. 255] \to 255.$$

Т.е., если значение пикселя входного изображения попадает в диапазон $[0 .. f_0]$, где f_0 - первый порог, то этот пиксель становится равным 0. Если же пиксель попал в $[f_1 + 1 .. f_2]$, то в выходное изображение вместо него записывается 170.

Файл, содержащий функцию `main` должен называться `hard.cpp` или `hard.c`. Если этого не сделать, то автотесты не пройдут, т.к. будет непонятно, какой вариант вы выполнили.

Помимо времени (см. Задание) необходимо выводить значения найденных порогов в консоль. Формат вывода (в формате Си): “`%u %u %u\n`”. Порядок вывода времени и порогов не важен.

Формат хранения изображения

Формат входных и выходных изображений: PNM (P5). Во входных и выходных PNM файлах комментарии отсутствуют. Формат хранения:

```
P5\n<ширина> <высота>\n255\n<массив значений оттенков пикселей>
```

Требования к работе программы

1. Аргументы программе передаются через командную строку:
   `omp4 <кол-во_потоков> <имя_входного_файла> <имя_выходного_файла>`
   где omp4 - имя исполняемого файла (то есть это argv[0]).
   1.1. Число потоков может равняться -1 и больше. 0 соответствует значению числа потоков по умолчанию. -1 - запуск без openmp.
   1.2. На вход вы получаете изображение в формате PNM (см. раздел PNM).
   1.3. В выходной файл записывается результат работы программы в формате, указанном в каждом варианте.
2. Корректно выделяется и освобождается память, закрываются файлы, есть обработка ошибок: не удалось открыть файл, формат файла не поддерживается.
3. Если программе передано значение, которое не поддерживается – следует сообщить об ошибке.
4. В программе можно вызывать только стандартные библиотеки (например, <bits/stdc++.h> таковой не является и ее использование влечет за собой потерю баллов). То есть сторонние библиотеки использовать нельзя (библиотека <omp.h> и модули для подключения openmp конечно разрешены).
5. Если программа использует библиотеки, которые явно не указаны в файле с исходным кодом (например, <algorithm>), то за это также будут снижаться баллы.

Оформление в отчёте

1. **Никаких скринов кода – код в отчет добавляется только текстом!**
2. Шрифт: `Consolas` (размер 10-14 на ваше усмотрение).
3. Выравнивание по левому краю.
4. Подсветка кода допустима. Текст должен быть читаемым (а не светло-серый текст, который без выделения на белом не разобрать).
5. В раздел Листинг код вставляется полностью в следующем виде:

<Название файла>

<Его содержимое>

Файлы исходных кодов разделяются новой строкой.

Например,

main.cpp

```
int main()
{
   return 0;
}
```

tmain.cpp

```
int tmain()
{
   return 666;
}
```

6. Фон белый (актуально для тех, у кого копипаста кода идет вместе с фоном темной темы из IDE).

Github и Github Actions

В общем об Actions можно почитать по ссылке ЭВМ осень 2022. Github (AT).

Как и в ЛР2 Readme автообновляемый автотестами и состоит из следующих разделов (выводится лог с windows):

1. Статус последнего теста (badge)
2. Build (сборка решения: `clang++ -std=c++20 -D _CRT_SECURE_NO_WARNINGS -D _USE_MATH_DEFINES -fopenmp -O2 <все *c и *.cpp файлы во всех подпапках репозитория> -o <execute file>`)
3. Output (вывод вашей программы; может отсутствовать, если сборка не удалась).

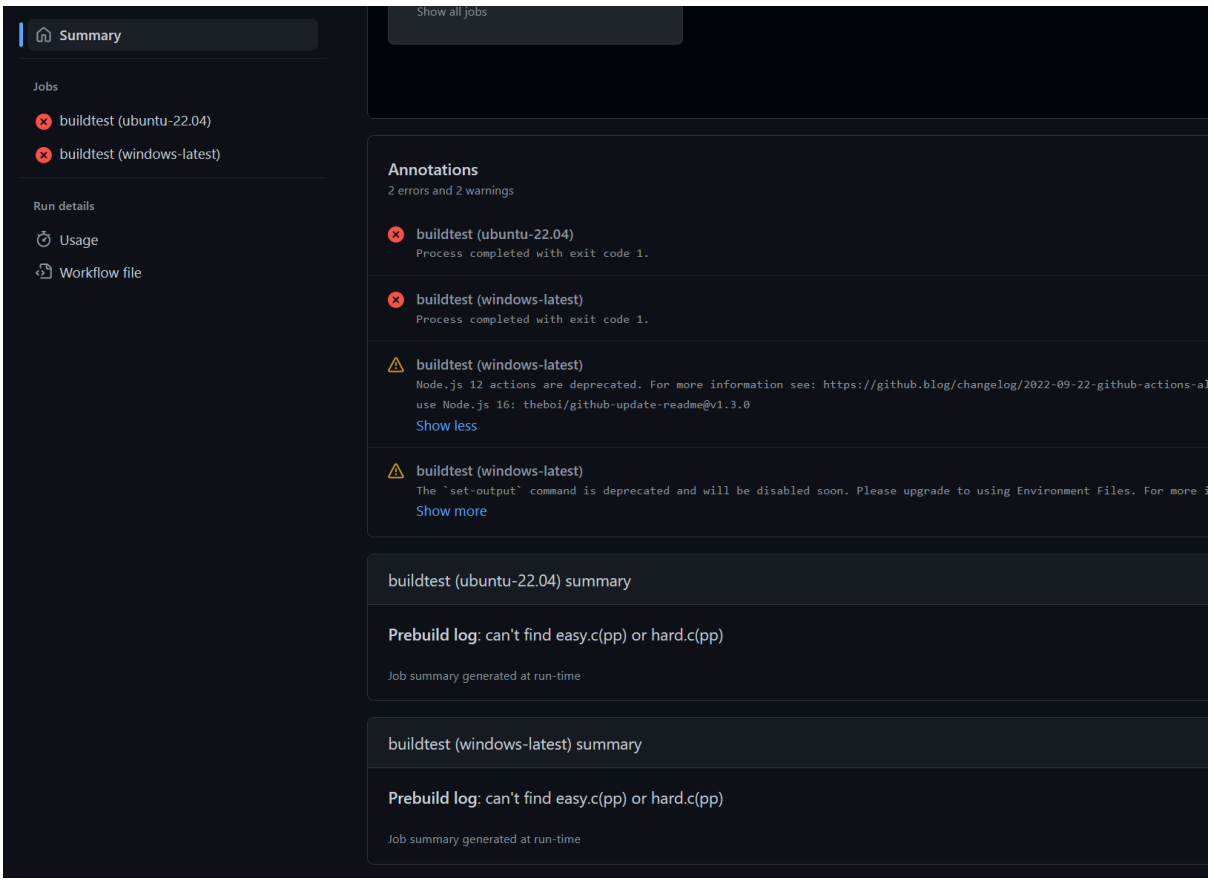
Если компиляция/линковка завершены неудачно, то статус автотеста будет failed. Если же всё собралось и программа запускается, но падает, то также будет failed и в разделе Output будет лог с ошибкой. В случае, если выходной файл не создан или программа что-то вывела в консоль, то тест будет завершен со статусом passed и дальше вы уже самостоятельно должны проверить в разделе Output, что вывод ровно такой, какой вы хотели.

Так вы сможете проверить, что ваш код работает не только у вас.

В репозиторий категорически запрещено заливать exe и отладочные файлы IDE, генерируемые самими IDE. Для этого у вас настроен .gitignore, поэтому удалять этот файл запрещено. Заливать <project>.sln или cmakefile можно, но они никак не будут использоваться при сборке решения при проверке и на автотестах.

Посмотреть логи с windows и ubuntu можно в summary последнего запуска: Actions - последний запуск - summary (пример ниже). У логов в summary может немного съехать оформление, но суть должна быть

понятна. Если тесты не будут работать или будут выдавать непонятные сообщения - пишем Виктории.



Полезное

C/C++

Для тех, кто не знает C/C++, предлагается ознакомится с содержимым файла ".github/workflows/example.cpp", в котором приведён код небольшой программы. В нём показано, как читать из файла, создавать массивы данных, проходиться по нему и что-то записывать в выходной файл с освобождением памяти программы.

Синтаксис языка и полезные функции можно посмотреть на следующем сайте: cppreference.com

Компиляция

Для включения OpenMP нужно указать ключ компиляции.

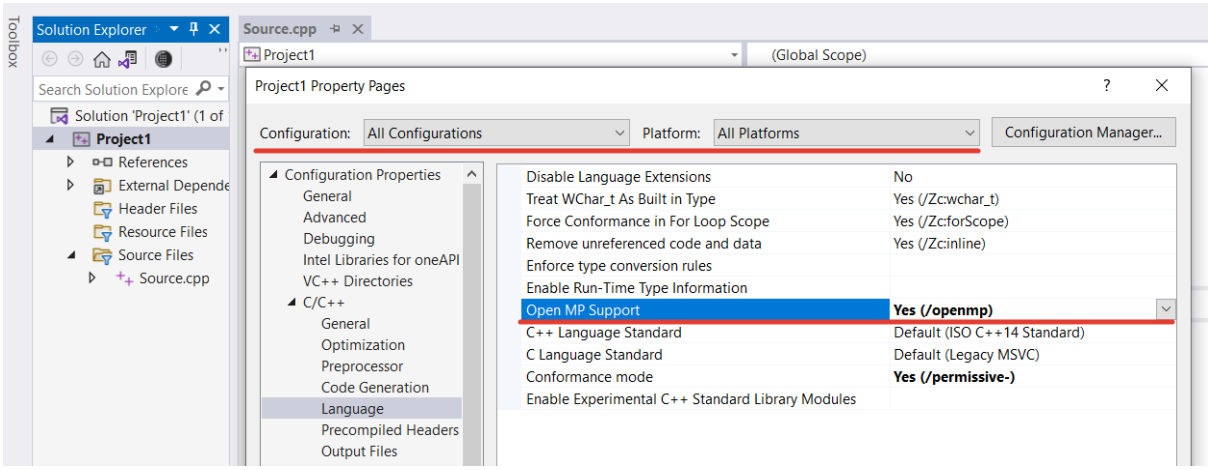
Компиляция через командную строку

Компилятор	Ключ компиляции
msvc (компилятор от Microsoft Visual Studio)	/openmp
gcc и clang	-fopenmp

Примечание: clang, который установлен на MacOS по умолчанию, может не содержать OpenMP. Нужно либо установить полный clang, либо поставить gcc. Подробнее можно посмотреть здесь: https://www.programmersought.com/article/93289356924/

Visual Studio

Свойства проекта (ПКМ по проекту в обозревателе проектов) - C/C++ - Language - OpenMP support - Yes.



CMakeLists

```
OPTION (USE_OpenMP "Use OpenMP" ON)
IF(USE_OpenMP)
    FIND_PACKAGE(OpenMP)
    IF(OPENMP_FOUND)
        SET(CMAKE_C_FLAGS "${CMAKE_C_FLAGS} ${OpenMP_C_FLAGS}")
        SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} ${OpenMP_CXX_FLAGS}")
    ENDIF()
ENDIF()
```

Для владельцев Mac на ARM64 также может помочь

```
set(CMAKE_OSX_ARCHITECTURES x86_64)
```